**Unit tests**

**JEST**

Test(‘add 2 + 2 to equal 4’, () => {

Expect(add(2,2)).toBe(4);

})

Basic methods

|  |  |  |
| --- | --- | --- |
| toBe(*result)* | not.toBe(*result)* | toBeNull() |
| not.toBeNull() | toBeFalsy() | toBeTruthy() |
| toEqual(*result)* | toBeLessThan(result) | toBeLessThanOrEqual(*result)* |
| toMatch(*regex)* | toContain(*result)* | toBeCalled() |

Working with promises

Test(‘user fetched name should be Ali’, () => {

Expect.assertions(1); // number of assertions (important)

Return fetchUser()

.then(data => {

Expect(data.name).toEqual(‘Ali’);

})

})

Async/Await

Test(‘user to be Ali’, async () => {

Expect.assertions(1);

Const data = await fetchUser();

Expect(data.name).toEqual(‘Ali’);

})

DOM manipulation

Test(‘test HTML exists’, () => {

Document.body.innerHTML = ‘<div id=”testEventListener”></div>’;

Expect(*functionName).*toBeCalled();

})

Mock functions

Const mockFunction = jest.fn();

All mocks must have a ‘mock’ property which has data about how the function has been called.

Const add = jest.fn(() => 3); // the add mocked function will always return 3

Setup and teardown

If you need to repeat the same code for your tests you can do so in jest built-in functions instead of repeating code:

beforeEach(() => {

// code to do before each test

})

// use afterEach(); for after tests teardowns

// There is also a beforeAll() and afterAll()

**PHPUNIT**

Assertions

|  |  |  |
| --- | --- | --- |
| assertArrayHasKey() | assertContains() | assertCount() |
| assertEmpty() | assertEquals() | assertFalse() |
| assertFileExists() | assertGreaterThan() | assertGreaterThanOrEqual() |
| assertInstanceOf() | assertIsArray() | assertIsBool() |
| assertIsFloat() | assertIsInt() | assertIsCallable() |
| assertIsIterable() | assertIsNumeric() | assertIsObject() |
| assertIsScalar() | assertIsString() | assertSame() |
| assertNan() | assertNull() |  |

Testing exceptions

$this->expectException(InvalidArgumentException::class);

// you can use Errro::class to test for standard PHP errors

Setup & teardown

Setup() – this is invoked before the test methods of the class it relates to, it is in this method that you create objects against which you will test

Teardown() – invoked after test finished. This is where you can clean up the objects which you tested

Incomplete tests

$this->markTestIncomplete(‘test needs implementing’);

Test doubles

**Stubs** – The practice of replacing an object with a test double that (optionally) returns configured return values

**Mock –** the practice of replacing an object with a test double that verified expectations, for instance asserting that a method was called

**Spies –** The main difference between spies and the other mock objects, is that with spies, we verify the calls made against our test double after the calls were made

Mockery

Mockery::mock(‘MyClass’); // stubs and mocks

Mockery::spies(‘MyClass’); // spy

e.g.

$mock = \Mockery::mock(‘MyClass’);

$spy = \Mockery::spies(‘MyClass’);

$mock->shouldReceive(‘foo’)->andReturn(42); // expect before call

$mockResult = $mock->foo(); // call

$spyResult = $spy->foo(); // call

$spy->shouldHaveReceived()->foo(); // expect after call

Var\_dump($mock); // int(42)

Var\_dump($spy); //null 🡨 always null for a spy

Declaring method call expectations

$mock->shouldReceive(‘methodName’)->andReturn(true);

$mock->shouldReceive(‘methodName’)->with($arg1, $arg2)->andReturn(true);

$mock->shouldReceive(‘method1’, ‘method2’)-andReturn(true); // all methods return true

$mock->shouldReceive([

‘method1’ => ‘returned value’,

‘method2’ => ‘returned value’,

]);

Throwing exceptions

We can tell the method of mock objects to throw exceptions:

$mock->shouldReceive(‘methodName’)->andThrow(\Exception);